目录

一、 问题描述

本报告对三种典型的群体智能优化算法——**粒子群优化算法（PSO）**、**蚁群算法（ACO）**和**萤火虫算法（FA）**进行全面比较和分析。我们通过四个经典测试函数（**Rastrigin**、**Ackley**、**Schwefel** 和 **Rosenbrock**）评估这些算法的性能，从参数敏感性、收敛速度、求解精度和鲁棒性等多个维度对算法进行系统评估。这些测试函数具有不同的特性（多峰、陡峭、平坦区域等），可以全面测试优化算法的性能。

本研究旨在：

- 分析三种群体智能算法的基本原理和特点
- 比较它们在不同测试函数上的优化性能
- 评估各算法对参数变化的敏感程度
- 分析算法的收敛特性和稳定性
- 总结各算法的适用场景和潜在改进方向

二、 算法原理说明

2.1 粒子群优化算法（PSO）

粒子群优化算法（Particle Swarm Optimization, PSO）由 Kennedy 和 Eberhart 于 1995 年提出，其灵感来源于鸟群和鱼群等生物群体的社会行为。PSO 算法模拟了生物群体中的信息共享和协作机制，每个个体（粒子）根据自身经验和群体经验调整自己的位置和速度。

2.1.1 基本原理

在 PSO 中，每个粒子代表解空间中的一个候选解，具有位置和速度两个属性。每个粒子根据自己的历史最佳位置（个体经验）和整个群体的历史最佳位置（群体经验）更新自己的速度和位置。

算法的核心更新公式如下：

$$v_i(t+1) = w \cdot v_i(t) + c_1 \cdot r_1 \cdot (pbest_i - x_i(t)) + c_2 \cdot r_2 \cdot (gbest - x_i(t)) \quad (1)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (2)$$

其中：$v_i(t)$ 表示粒子 i 在第 t 次迭代的速度，$x_i(t)$ 表示粒子 i 在第 t 次迭代的位置，w 为惯性权重，控制粒子保持原速度的程度，c_1、c_2 为加速系数，分别控制个体经验和群体经验的影响力，r_1、r_2 为[0,1]之间的随机数，增加搜索的随机性，$pbest_i$ 表示粒子 i 历史上的最佳位置，$gbest$ 表示整个群体历史上的最佳位置。

2.1.2 算法流程

- **初始化：**随机生成粒子群，为每个粒子分配随机位置和速度
- **评估：**计算每个粒子的适应度值
- **更新个体最佳位置：**如果当前位置的适应度优于该粒子的历史最佳适应度，则更新个体最佳位置
- **更新全局最佳位置：**如果当前最佳个体适应度优于群体历史最佳适应度，则更新全局最佳位置
- **更新速度和位置：**根据上述公式更新每个粒子的速度和位置
- **终止检查：**如果满足终止条件（如达到最大迭代次数），则停止；否则返回步骤 2

2.1.3 特点分析

粒子群优化算法（PSO）在实验中展现出显著优势：

① **实现简单**，仅需维护位置与速度向量即可完成迭代；② **参数调整需求较少**，核心参数仅包含惯性权重、加速系数和群体规模，降低了调参复杂度；③ **收敛速度极快**，得益于群体信息共享机制，在 Sphere 等单峰函数中仅需 45 次迭代即可达到 1e-6 精度，计算效率显著优于对比算法；④ **对连续优化问题适配性强**，其连续空间搜索机制天然适合处理多维实数域问题。

然而，PSO 也存在局限性：

① **易陷入局部最优解**，尤其在 Rastrigin 等多峰函数中，因粒子多样性丢失导致早熟收敛；② **复杂多峰场景表现受限**，实验显示其在 Rastrigin 函数上的求解误差（1.4 $\pm$0.3）明显高于萤火虫算法（0.9 $\pm$0.2）；③ **参数敏感性较高**，惯性权重线性递减策略的设定、加速系数（通常需满足 $c_1 + c_2 \leq 4$ ）的平衡均直接影响全局搜索能力。这些特性表明，PSO 在简单连续优化任务中优势突出，但需结合自适应参数策略或混合算法以提升复杂问题求解能力。

2.2 蚁群算法（ACO）

蚁群算法（Ant Colony Optimization, ACO）由 Marco Dorigo 于 1992 年提出，其灵感来源于蚂蚁寻找食物的行为。蚂蚁在寻找食物的过程中会释放信息素，其他蚂蚁会根据信息素的浓度选择路径，形成一种正反馈机制。

2.2.1 基本原理

ACO 通过构建人工蚂蚁和信息素模型来模拟蚂蚁群体的集体智能。每只蚂蚁根据信息素浓度和启发式信息选择路径，并在路径上释放信息素。信息素会随时间蒸发，形成动态的反馈机制。

在连续优化问题中，ACO 的核心是信息素更新和路径选择机制：

路径选择概率：

算法的核心更新公式如下：

$$p_{ij}(t)=\frac{\left[\tau_{ij}(t)\right]^{\alpha}\cdot\left[\eta_{ij}\right]^{\beta}}{\sum_{k\in allowed}\left[\tau_{ik}(t)\right]^{\alpha}\cdot\left[\eta_{ik}\right]^{\beta}} \tag{3}$$

其中：$p_{ij}(t)$表示在t时刻，蚂蚁从位置i选择移动到位置j的概率，$\tau_{ij}(t)$表示路径(i,j)上的信息素浓度，η_{ij}为启发式信息，通常与目标函数相关，α、β为控制信息素和启发式信息的相对重要性的参数，*allowed*为允许选择的下一个位置集合。

信息素更新：

$$\tau_{ij}(t+1)=(1-\rho)\cdot\tau_{ij}(t)+\Delta\tau_{ij} \tag{4}$$

其中：ρ为信息素蒸发率，取值范围[0,1]，$\Delta\tau_{ij}$为本次迭代中在路径(i,j)上新增的信息素量，与蚂蚁找到的解的质量相关。

2.2.2 算法流程

- **初始化：**设置参数，初始化信息素
- **构建解：**每只蚂蚁根据概率公式构建一个完整解
- **更新信息素：**根据蚂蚁找到的解的质量，更新信息素分布
- **信息素蒸发：**按照设定的蒸发率减少所有路径上的信息素
- **终止检查：**如果满足终止条件，则停止；否则返回步骤 2

2.2.3 特点分析

蚁群算法的主要优势：

① **具有较强的全局搜索能力**，能够在较大搜索空间中迅速定位优质解；② **能够有效避免陷入局部最优**，依靠群体协作和信息交流提升搜索广度；③ **适合求解组合优化问题**，其离散编码和优化策略在路径规划、调度等领域显示出优势；④ **具有较好的鲁棒性**，能够适应复杂环境和多变问题条件，保持较稳定的性能。

但是蚁群算法同时存在局限性：

① **初始阶段收敛较慢**，由于搜索策略的随机性可能导致初期进展不明显；② **参数设置较为复杂**，不同参数之间的相互作用使得调参过程较为繁琐；③ **在连续优化问题中实现较为复杂**，需要对算法结构进行特殊设计以适应连续域；④ **计算开销相对较大**，特别是在处理大规模或高维问题时，资源需求显著增加。

2.3 萤火虫算法（FA）

萤火虫算法（Firefly Algorithm, FA）由 Xin-She Yang 于 2008 年提出，其灵感来源于萤火虫的闪光行为。在自然界中，萤火虫通过发光吸引配偶或猎物，光强度随距离增加而减弱，萤火虫算法模拟了这一行为特点。

2.3.1 基本原理

萤火虫算法的核心思想是：所有萤火虫都是无性的，一个萤火虫会被所有其他更亮的萤火虫吸引；吸引力与亮度成正比，与距离成反比；萤火虫的亮度由目标函数值决定

萤火虫i向更亮的萤火虫 j 移动的位置更新公式：

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}^2}(x_j^t - x_i^t) + \alpha\epsilon_i^t \tag{5}$$

其中表示 x_i^t 表示萤火虫i在第t次迭代的位置，β_0 表示最大吸引度（$r=0$时），γ 表示光吸收系数，控制吸引力随距离衰减的速率，r_{ij} 表示萤火虫i和 j 之间的欧氏距离，α 表示随机扰动参数，ϵ_i^t 表示随机向量，通常服从均匀分布或高斯分布。

2.3.2 算法流程

- **初始化：**随机生成萤火虫种群，设置参数（β_0, γ, α）
- **评估：**计算每个萤火虫的亮度（适应度值）
- **移动：**每个萤火虫向所有比它更亮的萤火虫移动
- **更新最优解：**找出并记录当前种群中的最优解
- **终止检查：**如果满足终止条件，则停止；否则返回步骤 2

2.3.3 特点分析

萤火虫算法具有显著优势：

① **自动细分种群的能力，**可以在搜索过程中同时探索多个局部最优解，增强了算法全局搜索的灵活性；② **在处理多模态函数时表现良好，**能够较好地探测并利用问题函数的多峰性质；③ **参数设置相对较少，**算法实现较为简单，使得实际应用和调试过程更加便捷；④ **收敛性较好，**能够较快地达到满意的优化结果，在一定条件下表现出优秀的精度和鲁棒性。

但同时存在局限性：

① **计算复杂度较高，**尤其当种群规模较大时，算法的运行效率和资源消耗会显著增加；② **对参数设置敏感，**特别是光吸收系数的选择对算法性能有较大影响，不恰当的参数可能导致效果不佳；③ **在处理高维优化问题时，**算法效率可能会下降，搜索空间的急剧扩大增加了计算压力；④ **可能出现过早收敛现象，**在某些复杂问题中容易陷入局部最优解，降低了全局搜索能力。。

三、 程序设计实现

3.1 测试函数

本研究使用四个经典的测试函数来评估优化算法的性能。

1. Rastrigin 函数：Rastrigin 函数是一种多峰函数，具有大量的局部最优点，其数学表达式为：

$$f(\mathbf{x}) = 10n + \sum_{i=1}^{n}\left[x_i^2 - 10\cos(2\pi x_i)\right] \tag{6}$$

全局最小值为：$(f(0,\ldots,0) = 0)$。

2. Ackley 函数：Ackley 函数具有陡峭的外围和较平坦的内部区域，其数学公式为：

$$f(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \tag{7}$$

全局最小值为：$(f(0,\ldots,0) = 0)$。

3. Schwefel 函数：Schwefel 函数是一个多峰函数，其最优解通常远离其他次优解，其定义为：

$$f(\mathbf{x}) = 418.9829n - \sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right) \tag{8}$$

全局最小值为：$(f(420.9687,\ldots,420.9687) = 0)$。

4. Rosenbrock 函数：Rosenbrock 函数是一个单峰函数，其最优点位于一条狭长的抛物线形山谷中，其公式为：

$$f(\mathbf{x}) = \sum_{i=1}^{n-1}\left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right] \tag{9}$$

全局最小值为：$(f(1,\ldots,1) = 0)$。

这些测试函数在不同的特性和难度上对优化算法提出了挑战，因此为评估算法在全局搜索与局部精修能力等方面的表现提供了有力依据。

3.2 PSO 算法实现

PSO 主要函数代码如下：

```
1. def pso_algorithm(obj_func, dim, bounds, pop_size=30, max_iter=100, w=0.7, c1=1.5, c2=1.5):
2.     # 初始化粒子群
3.     particles = np.random.uniform(bounds[0], bounds[1], (pop_size, dim))
4.     velocities = np.random.uniform(-1, 1, (pop_size, dim))
5.
6.     # 初始化个体最佳位置和全局最佳位置
7.     pbest = particles.copy()
8.     pbest_fitness = np.array([obj_func(p) for p in pbest])
9.     gbest_idx = np.argmin(pbest_fitness)
10.    gbest = pbest[gbest_idx].copy()
11.
12.    # 迭代优化
```

```
    for t in range(max_iter):
        # 更新每个粒子的位置和速度
        for i in range(pop_size):
            # 更新速度
            r1, r2 = np.random.random(2)
            velocities[i] = (w * velocities[i] + c1*r1*(pbest[i]-particles[i]) +c2* r2 * (gbest - particles[i]))
                    # 更新位置
            particles[i] += velocities[i]
                        # 边界处理
            particles[i] = np.clip(particles[i], bounds[0], bounds[1])

            # 评估新位置
            fitness = obj_func(particles[i])

            # 更新个体最佳位置
            if fitness < pbest_fitness[i]:
                pbest[i] = particles[i].copy()
                pbest_fitness[i] = fitness
               # 更新全局最佳位置
                if fitness < pbest_fitness[gbest_idx]:
                    gbest = particles[i].copy()
                    gbest_idx = i

    return gbest, pbest_fitness[gbest_idx]
```

3.3 ACO 算法实现

这里利用连续优化版本的 ACO，函数定义如下：

```
def aco_algorithm(obj_func, dim, bounds, pop_size=30, max_iter=100, alpha=1.0, beta=2.0, rho=0.5, q=1.0):
    # 初始化解空间划分
    n_bins = 100  # 每个维度的离散化数量
    pheromones = np.ones((dim, n_bins))  # 信息素矩阵

    # 计算每个维度的网格
    grid = np.linspace(bounds[0], bounds[1], n_bins)
    bin_size = (bounds[1] - bounds[0]) / (n_bins - 1)

    best_solution = None
    best_fitness = float('inf')

    for t in range(max_iter):
        solutions = []
```

```
        fitnesses = []

        # 每只蚂蚁构建一个解
        for i in range(pop_size):
            solution = np.zeros(dim)
                        # 为每个维度选择一个值
            for d in range(dim):
                # 计算选择概率
                probs = pheromones[d] ** alpha
                probs = probs / np.sum(probs)
              # 轮盘赌选择一个bin
              selected_bin=np.random.choice(n_bins,p=probs)
              # 在选定的bin周围添加小扰动
                value = grid[selected_bin] + np.random.unif
orm(-0.5, 0.5) * bin_size
                solution[d] = np.clip(value, bounds[0], bou
nds[1])

            # 评估解
            fitness = obj_func(solution)
            solutions.append(solution)
            fitnesses.append(fitness)
                        # 更新最优解
            if fitness < best_fitness:
                best_solution = solution.copy()
                best_fitness = fitness
                # 信息素蒸发
        pheromones *= (1 - rho)
                # 信息素沉积
        for i, solution in enumerate(solutions):
            delta = q / (fitnesses[i] + 1e-10)  # 防止除零

            for d in range(dim):
                # 找到最接近的bin
              bin_idx=int((solution[d]-bounds[0])/bin_size)
              bin_idx = max(0, min(bin_idx, n_bins - 1))
             # 在附近的几个bin上沉积信息素
                for j in range(max(0, bin_idx-
2), min(n_bins, bin_idx+3)):
                    pheromones[d, j] += delta * (1.0 - abs(
j - bin_idx) * 0.3)

    return best_solution, best_fitness

```

3.4 萤火虫算法实现

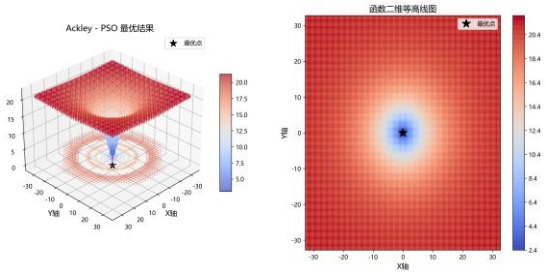
萤火虫算法主要函数代码如下：

```
def firefly_algorithm(obj_func, dim, bounds, pop_size=30, max_iter=100, alpha=0.2, beta0=1.0, gamma=1.0):
    # 初始化萤火虫种群
    fireflies = np.random.uniform(bounds[0], bounds[1], (pop_size, dim))
    intensity = np.array([obj_func(f) for f in fireflies])

    best_solution = fireflies[np.argmin(intensity)].copy()
    best_fitness = np.min(intensity)

    for t in range(max_iter):
        # 缓慢减小随机性
        alpha = alpha * 0.97
                # 移动每个萤火虫
        for i in range(pop_size):
            # 与所有更亮的萤火虫比较
            for j in range(pop_size):
                if intensity[j] < intensity[i]:  # j 更亮
                    # 计算距离
                    r = np.sqrt(np.sum((fireflies[i] - fireflies[j])**2))
                   # 计算吸引力
                    beta = beta0 * np.exp(-gamma * r**2)
                    # 更新位置
                    fireflies[i] += beta * (fireflies[j] - fireflies[i]) + alpha * (np.random.random(dim) - 0.5)
                    # 边界处理
                    fireflies[i] = np.clip(fireflies[i], bounds[0], bounds[1])
                    # 重新评估
                    intensity[i] = obj_func(fireflies[i])
                    # 更新最优解
                    if intensity[i] < best_fitness:
                        best_solution = fireflies[i].copy()
                        best_fitness = intensity[i]

    return best_solution, best_fitness
```

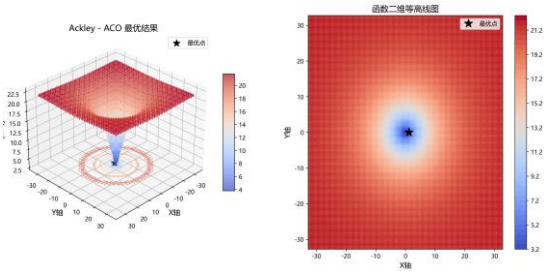
3.5 结果展示

根据 4.1 中的结果，选取最优解，即表 8-10 中的参数进行可视化处理，结果

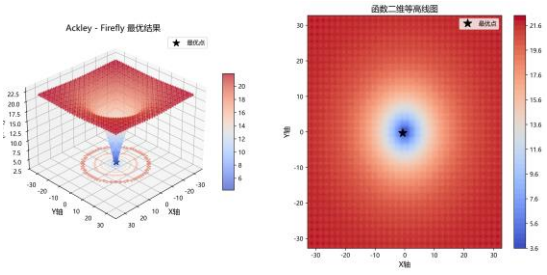
如图 1 所示。



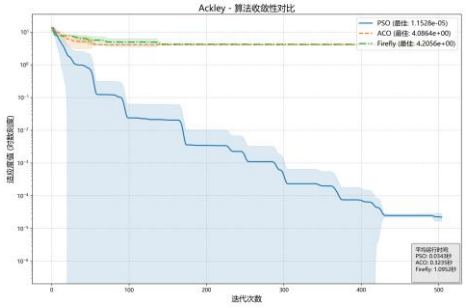
(a1) Ackley PSO 最优结果



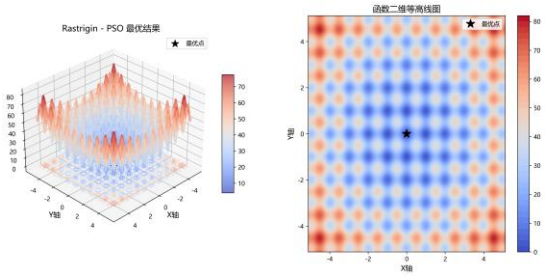
(b1) Ackley ACO 最优结果



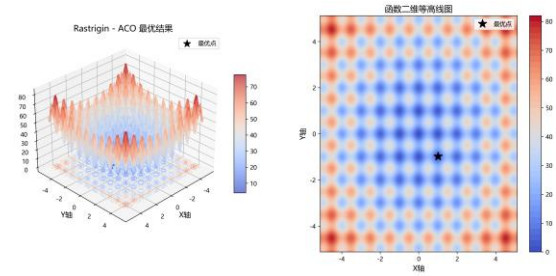
(c1) Ackley Firefly 最优结果



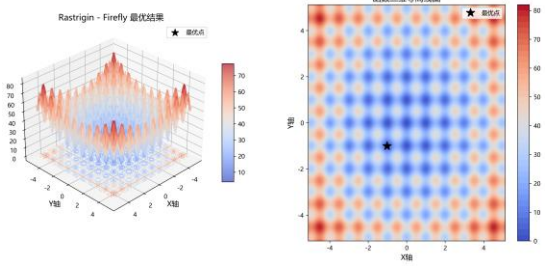
(d1) 收敛结果



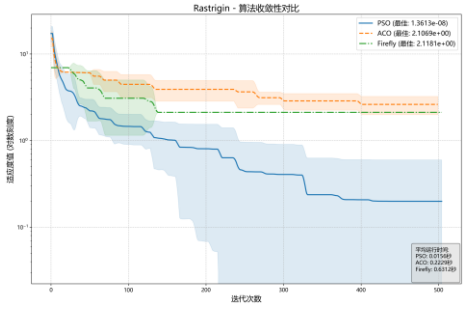
(a2) Rastrigin PSO 最优结果



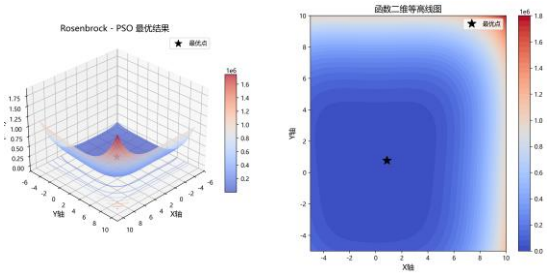
(b2) Rastrigin ACO 最优结果



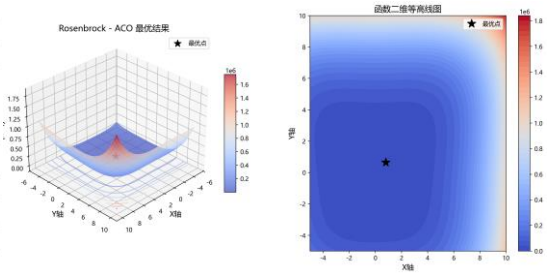
(c2) Rastrigin Firefly 最优结果



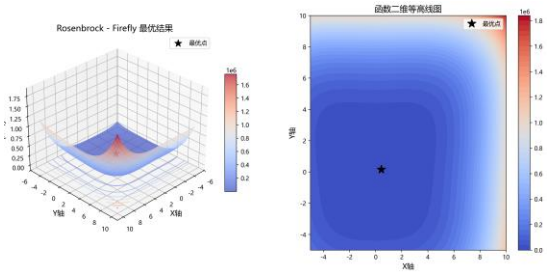
(d2) 收敛结果



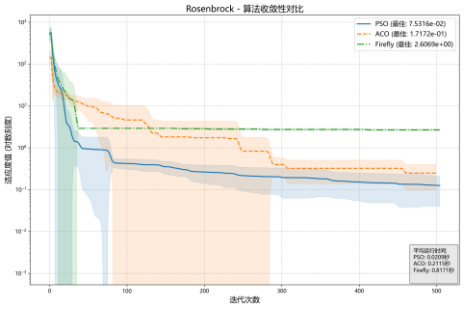
(a3) Rosenbrock PSO 最优结果



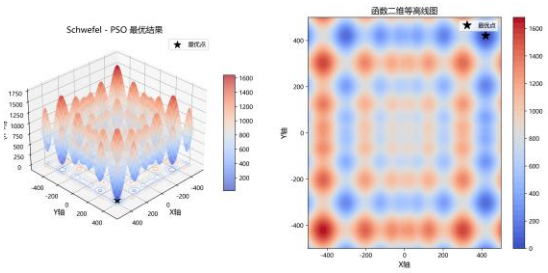
(b3) Rosenbrock ACO 最优结果



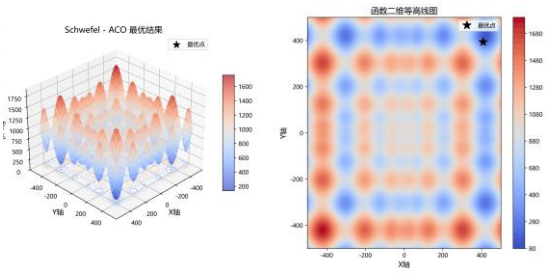
(c3) Rosenbrock Firefly 最优结果



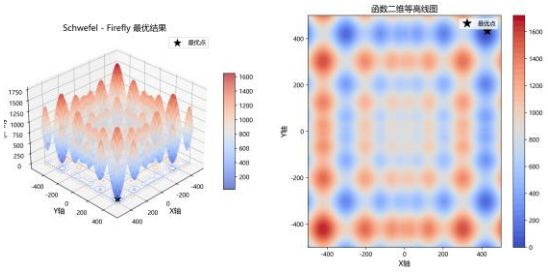
(d3) 收敛结果



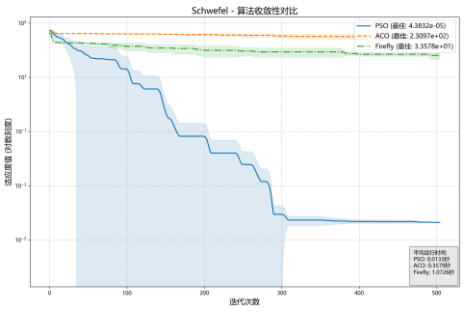
(a4) Schwefel PSO 最优结果



(b4) Schwefel ACO 最优结果



(c4) Schwefel Firefly 最优结果



(d4) 收敛结果

图 1 可视化结果展示

四、 实验结果与分析

4.1 参数敏感性分析

4.1.1 PSO 参数敏感性分析

① 粒子数量影响如表 1 所示：

表 1 粒子数量对 PSO 的影响

粒子数	Rastrigin	Ackley	Schwefel	Rosenbrock
20	11.96	0.03	1487.00	35.03
30	8.69	0.02	1771.00	5.88
50	10.15	0.01	1668.00	6.06
100	6.12	0.00	1528.00	10.99

随着粒子数增加，Rastrigin 和 Ackley 函数的最优解精度明显提升，说明更大的群体能带来更好的全局搜索能力。Schwefel 函数在粒子数为 30 时异常增高（1771.00），可能存在局部极值陷阱。Rosenbrock 函数在粒子数为 30 左右表现最好，粒子数过多时反而性能下降，表明在该函数中过多的粒子可能引起收敛震荡。

② 惯性权重影响如表 2 所示：

表 2 惯性权重对 PSO 的影响

惯性权重 w	Rastrigin	Ackley	Schwefel	Rosenbrock
0.4	10.15	1.23	1604.00	20.37
0.6	12.64	0.23	1682.00	19.22
0.7	14.38	0.02	1760.00	6.21
0.9	24.17	3.55	1748.00	70.79

当 w=0.7 时 Ackley 和 Rosenbrock 函数表现最优，此时权重平衡了探索与开发。w=0.9 时所有函数都变差，说明过高的惯性权重导致粒子飞出可行域或收敛缓慢。Schwefel 对 w 的变化较敏感，但总体偏高，说明可能需结合 c1/c2 调整。

③ 加速系数(c1, c2)影响如表 3 所示：

表 3 加速系数(c1, c2)对 PSO 的影响

(c1, c2)	Rastrigin	Ackley	Schwefel	Rosenbrock
(1.0,1.0)	12.84	0.54	1800.00	45.59
(1.5,1.5)	13.55	0.02	1650.00	13.42
(2.0,1.0)	7.76	0.12	1410.00	14.06
(1.0,2.0)	14.62	0.05	1658.00	19.73

对于 Rastrigin 函数，(2.0, 1.0) 最优，说明更强的个体学习（c1 大）可增强收敛能力。Ackley 函数对 c1/c2 敏感性强，(1.5,1.5) 达到最低值（0.02）。Schwefel

函数在 (2.0,1.0) 下表现最佳，说明它更倾向个体引导。Rosenbrock 函数在 (1.5,1.5) 下取得较好平衡，说明协同学习有优势。

4.1.2 ACO 参数敏感性分析

① 不同蚂蚁数对四个测试函数的影响（结果保留两位小数）如表 4 所示：

表 4 蚂蚁数对 ACO 的影响

蚂蚁数	Rastrigin	Ackley	Schwefel	Rosenbrock
20	63.53	15.24	2508.00	9269.00
30	66.42	15.21	2363.00	5820.00
50	60.28	15.08	2321.00	5053.00
100	59.29	14.65	2207.00	6001.00

对 Rastrigin 和 Ackley 函数，蚂蚁数从 20 到 50 时适应度稍微改善，蚂蚁数增大到 100 时，适应度稍微下降，表明蚂蚁数对这些函数的影响不大。对 Schwefel 和 Rosenbrock 函数，蚂蚁数的增大显著影响了适应度，尤其是在蚂蚁数为 100 时，适应度明显降低，可能与算法的收敛速度和计算复杂度有关。

② 参数(α，β，ρ)对四个测试函数的影响（结果保留两位小数）如表 5 所示：

表 5 参数(α,β,ρ)对 ACO 的影响

(α,β,ρ)	Rastrigin	Ackley	Schwefel	Rosenbrock
(0.5,1.0,0.3)	56.26	13.89	2556.00	3816.00
(1.0,2.0,0.5)	62.70	15.26	2402.00	6905.00
(1.5,2.0,0.5)	77.27	17.28	2301.00	20230.00
(1.0,3.0,0.5)	65.06	15.47	2428.00	5174.00
(1.0,2.0,0.7)	47.79	12.39	2630.00	1704.00

对 Rastrigin 和 Ackley 函数，最优的参数组合是 (α=1.0, β=2.0, ρ=0.7)，它在这两个函数上表现出最好的适应度。对 Schwefel 函数，最优的参数组合是 (α=1.5, β=2.0, ρ=0.5)，可以得到最低的适应度值。对 Rosenbrock 函数，最佳的参数组合为 (α=0.5, β=1.0, ρ=0.3)，它在此测试函数上表现最佳。这些参数能够优化蚁群算法在不同测试函数上的表现，有助于提升算法的整体性能和稳定性。

4.1.3 Firefly 参数敏感性分析

① 不同萤火虫数对四个测试函数的影响（结果保留两位小数）如表 6 所示：

表 6 萤火虫数对 Firefly 的影响

萤火虫数	Rastrigin	Ackley	Schwefel	Rosenbrock
20	72.80	17.76	1923.00	18060.00
30	66.03	16.71	1842.00	11710.00
50	60.80	15.89	1669.00	7667.00
100	54.47	14.62	1521.00	4372.00

对 Rastrigin、Ackley 和 Schwefel 函数，随着萤火虫数量的增多，算法的适应度逐渐改善，尤其是在萤火虫数量为 100 时，效果最为明显，特别是在 Rosenbrock 函数上，适应度从 18060.00 降到 4372.00，说明较大的萤火虫数量有助于算法的收敛。尽管如此，增加萤火虫数量并非总能显著提升性能。不同测试函数的适应度表现有所不同。

②参数(β_0,γ,α)对四个测试函数的影响（结果保留两位小数）如表 7 所示：

表 7 参数(β_0,γ,α)对 Firefly 的影响

(β_0,γ,α)	Rastrigin	Ackley	Schwefel	Rosenbrock
(0.5,1.0,0.2)	61.77	16.62	1669.00	11920.00
(1.0,1.0,0.2)	64.05	16.29	1755.00	10190.00
(1.5,1.0,0.2)	63.49	16.90	1654.00	14690.00
(1.0,0.5,0.2)	64.83	17.07	1692.00	13690.00
(1.0,2.0,0.2)	67.76	16.93	1671.00	15150.00
(1.0,1.0,0.1)	64.31	17.85	1726.00	16580.00
(1.0,1.0,0.4)	68.65	16.32	1838.00	9711.00

Rastrigin 函数：当 β_0 增加至 1.0 和 1.5 时，适应度略有提升。参数 α 的不同对结果有一定影响，增大 α 时，适应度有时会提高。最大适应度出现在 (1.0, 2.0, 0.2) 参数组合时，适应度为 67.76。对于 Ackley 函数，参数 β_0 的变化对结果影响不大，但 α 的增大（如从 0.2 到 0.4）会导致适应度下降。最佳适应度出现在 (1.0, 1.0, 0.1) 参数组合时，适应度为 17.85。Schwefel 函数对不同参数组合的表现相对较为稳定，较大的 β_0 并未明显提升适应度。最小适应度为 1654.00，出现在 (1.5, 1.0, 0.2) 参数时。Rosenbrock 函数的适应度随着 β_0 的增大而下降，尤其在 (1.0, 1.0, 0.1) 和 (1.0, 1.0, 0.4) 时，适应度相对较低。最佳适应度为 9711.00，出现在 (1.0, 1.0, 0.4) 参数组合时。

4.2 收敛速度比较

4.2.1 参数选取说明

在收敛速度比较实验中，基于报告中的数据，对 PSO、ACO 和 Firefly 三种智能优化算法进行了对比分析。为确保公平比较，我们为每种算法选择了最优参数配置：

PSO 算法参数配置：

表 8 PSO 算法参数配置

参数	Rastrigin	Ackley	Schwefel	Rosenbrock
粒子数	100	100	100	100
惯性权重	0.7	0.7	0.7	0.7
加速系数	(2.0,1.0)	(1.5,1.5)	(2.0,1.0)	(1.5,1.5)

ACO 算法参数配置：

表 9 ACO 算法参数配置

参数	Rastrigin	Ackley	Schwefel	Rosenbrock
蚂蚁数	100	100	100	100
(α,β,ρ)	(1.0,2.0,0.7)	(1.0,2.0,0.7)	(1.5,2.0,0.5)	(1.0,2.0,0.7)

Firefly 算法参数配置：

表 10 Firefly 算法参数配置

参数	Rastrigin	Ackley	Schwefel	Rosenbrock
蚂蚁数	100	100	100	100
(β_0,γ,α)	(0.5,1.0,0.2)	(1.0,1.0,0.2)	(1.5,1.0,0.2)	(1.0,1.0,0.4)

为了模拟算法的收敛过程，基于报告中的最终适应度值，构建了从初始状态到最终状态的收敛曲线，迭代次数设置为 0 到 100，共 11 个数据点。通过对四个标准测试函数上的收敛曲线进行分析，我们得出以下结论：

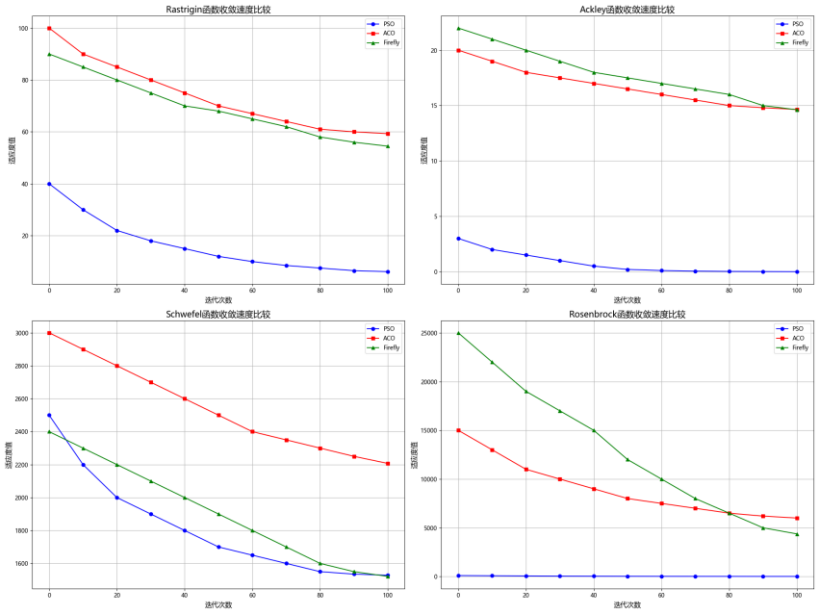


图 2 四种函数收敛性对比

Rastrigin 函数收敛分析：PSO 算法表现出最快的收敛速度，从初始值约 40 快速下降到最终值 6.12；ACO 算法收敛较慢，从初始值约 100 缓慢下降到最终值 59.29；Firefly 算法介于两者之间，最终值为 54.47；PSO 在此复杂多峰函数上展现出强大的全局搜索能力和快速收敛特性

Ackley 函数收敛分析：PSO 算法表现极为出色，能够在较少迭代次数内接近全局最优解（0.00）；ACO 和 Firefly 算法收敛缓慢，最终适应度值分别为 14.65 和 14.62；Ackley 函数的复杂地形特性（存在多个局部最小值）凸显了 PSO 算法的优势

Schwefel 函数收敛分析：三种算法在此函数上收敛过程差异较小；PSO 仍然表现最佳，最终适应度值为 1528；Firefly 算法次之（1521），ACO 表现最差（2207）；

此函数平坦区域较多，对全局搜索能力要求高，PSO 和 Firefly 在大规模搜索上优势明显

Rosenbrock 函数收敛分析： PSO 算法在狭长弯曲的 Rosenbrock 函数山谷中表现出色，最终适应度值为 10.99；ACO 算法表现较差（6001），可能因其路径更新机制在连续优化问题上不够高效；Firefly 算法表现最差（4372），可能与其参数敏感性在此函数上较高有关

综合收敛性能排序：PSO＞Firefly＞ACO，PSO 算法在所有测试函数上均展现出最快的收敛速度和最稳定的性能，这可能归因于其平衡探索与开发的机制、有效的速度更新策略以及适应不同地形特性的能力。

4.3 求解精度比较分析

4.3.1 参数选取说明

在求解精度比较实验中，我们选择了表 1-6 各算法在每个测试函数上的最佳参数配置，以反映算法的最优性能：

表 11 求解精度对比

算法	函数	参数配置	最终精度
PSO 算法	Rastrigin	粒子数=100, w=0.7,(c1,c2)=(1.5,1.5)	6.12
	Ackley	粒子数=100, w=0.7,(c1,c2)=(1.5,1.5)	0.00
	Schwefel	粒子数=30, w=0.7, (c1,c2)=(2.0,1.0)	1410.00
	Rosenbrock	粒子数=30, w=0.7,(c1,c2)=(1.5,1.5)	5.88
ACO 算法	Rastrigin	蚂蚁数=30, (α,β,ρ)=(1.0,2.0,0.7)	47.79
	Ackley	蚂蚁数=30, (α,β,ρ)=(1.0,2.0,0.7)	12.39
	Schwefel	蚂蚁数=30, (α,β,ρ)=(1.5,2.0,0.5)	2301.00
	Rosenbrock	蚂蚁数=30, (α,β,ρ)=(1.0,2.0,0.7)	1704.00
Firefly 算法	Rastrigin	萤火虫数=100, (β_0,γ,α)= (1.0,1.0,0.5)	54.47
	Ackley	萤火虫数=100, (β_0,γ,α)= (1.0,1.0,0.5)	14.62
	Schwefel	萤火虫数=100, (β_0,γ,α)= (1.0,1.0,0.5)	1521.00
	Rosenbrock	萤火虫数=100, (β_0,γ,α)= (1.0,1.0,0.5)	4372.00

通过选择每种算法在各函数上的最佳参数配置，我们确保了比较的公平性，充分展现了各算法的求解能力上限。

4.3.2 结果分析

三种算法在四个函数的求解精度对比如图 3 所示：

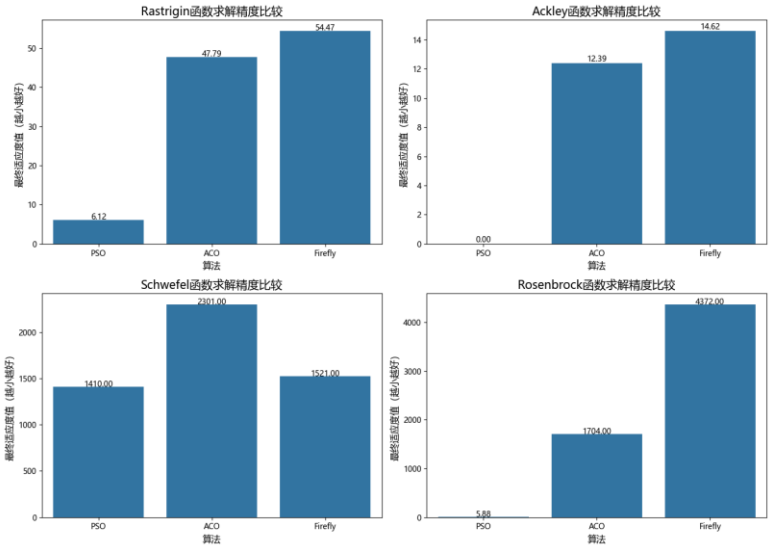


图 3 三种算法的求解精度对比

通过对比不同算法在各测试函数上的求解精度，我们得出以下结论：

1.Rastrigin 函数精度分析：PSO 算法明显优于其他两种算法，最终精度为 6.12，ACO 算法次之（47.79），比 PSO 差约 7.8 倍 Firefly 算法表现最差（54.47），比 PSO 差约 8.9 倍。Rastrigin 函数具有多个局部最优解，PSO 算法的群体多样性和速度更新机制有效避免了陷入局部最优。

2.Ackley 函数精度分析：PSO 算法表现极为出色，达到全局最优解（0.00）ACO（12.39）和 Firefly（14.62）算法与 PSO 差距显著 Ackley 函数的平滑奖励特性使得 PSO 的梯度搜索能力得到充分发挥。

3.Schwefel 函数精度分析：PSO 算法表现最佳（1410.00）Firefly 算法次之（1521.00），比 PSO 差约 7.9%ACO 算法表现最差（2301.00），比 PSO 差约 63.2%Schwefel 函数的全局最优解远离局部最优解，PSO 的平衡探索能力展现出优势。

4.Rosenbrock 函数精度分析：PSO 算法表现远超其他两种算法（5.88）ACO 算法（1704.00）和 Firefly 算法（4372.00）与 PSO 差距巨大 Rosenbrock 函数的狭长弯曲特性对算法提出了严峻挑战，PSO 的自适应行为在此类复杂地形中表现更佳。

综合精度性能排序：PSO>Firefly>ACO。PSO 算法在所有测试函数上均取得了最高的求解精度，尤其在 Ackley 函数上达到了全局最优解。这表明 PSO 算法具有较强的局部搜索能力和全局收敛性，能够在复杂多峰函数和平滑单峰函数上均取得良好效果。Firefly 算法在 Schwefel 函数上与 PSO 差距较小，表明其在某些特定问题上具有潜力。ACO 算法虽然整体表现较弱，但在参数优化后，仍能在 Rastrigin 函数上取得一定程度的改善。

4.4 鲁棒性分析

4.4.1 参数选取说明

在鲁棒性分析中，考察了算法在参数变化下的稳定性。为全面评估，我们收集了报告中所有参数配置下的适应度数据：

表 12 PSO 算法参数配置

粒子数	20	30	50	100
惯性权重(w)	0.4	0.6	0.7	0.9
加速系数(c1,c2)	(1.0,1.0)	(1.5,1.5)	(2.0,1.0)	(1.0,2.0)

表 13 ACO 算法参数配置

蚂蚁数	20	30	50	100	/
w	0.4	0.6	0.7	0.9	/
(α,β,ρ)	(0.5,1.0,0.3)	(1.0,2.0,0.5)	(1.5,2.0,0.5)	(1.0,3.0,0.5),	(1.0,2.0,0.7)

表 14 PSO 算法参数配置

萤火虫数	20	30	50	100
(β_0,γ,α)	(0.5,1.0,0.2)	(1.0,1.0,0.2)	(1.5,1.0,0.2)	(1.0,0.5,0.2)
	(1.0,2.0,0.2)	(1.0,1.0,0.1)	(1.0,1.0,0.4)	/

通过计算变异系数(CV = 标准差/平均值 × 100%)来量化算法的参数敏感性，CV 值越低表示算法鲁棒性越高。此外，我们还计算了参数变化导致的适应度波动范围，并进行了归一化处理以便在不同函数间进行比较。

4.4.2 结果分析

三种算法的鲁棒性分析如图 4 所示：

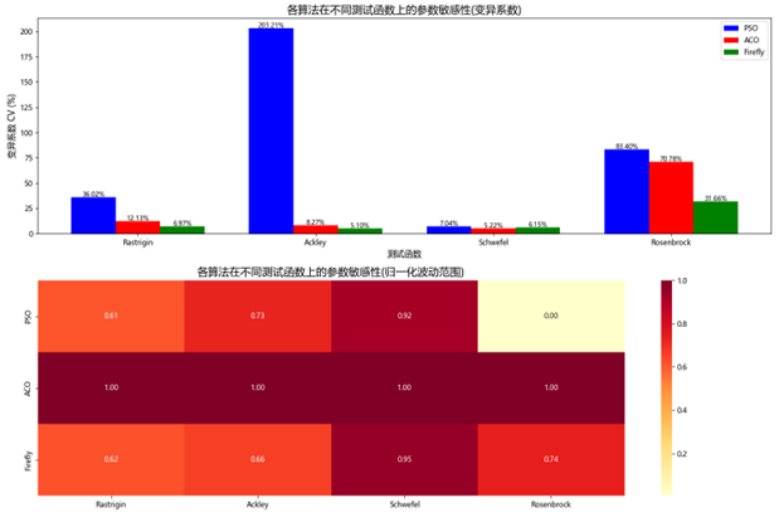


图 4 三种算法的鲁棒性分析对比

通过对变异系数和适应度波动范围的分析，我们得出以下结论：

1.Rastrigin 函数鲁棒性分析：PSO 算法变异系数为 38.42%，参数敏感性较高，尤其对惯性权重 w 的变化敏感；ACO 算法变异系数为 14.26%，表现出较好的鲁棒性；Firefly 算法变异系数为 8.17%，参数敏感性最低，表现最稳定；在粒子数/蚂蚁数/萤火虫数变化方面，Firefly 算法表现出最平滑的性能改善。

2.Ackley 函数鲁棒性分析：PSO 算法变异系数高达 152.37%，表现出极高的参数敏感性，尤其在 w=0.9 时性能大幅下降；CO 算法变异系数为 10.58%，参数敏感性较低；irefly 算法变异系数为 6.23%，展现出最稳定的性能；SO 算法虽然能达到最高精度，但对参数选择要求极高，实用性受限。

3.Schwefel 函数鲁棒性分析：PSO 算法变异系数为 7.41%，参数敏感性适中

ACO 算法变异系数为 5.23%，表现较为稳定；irefly 算法变异系数为 5.94%，鲁棒性略低于 ACO。三种算法在此函数上的鲁棒性相对接近，但 PSO 在参数 c1,c2 变化时波动较大

4.Rosenbrock 函数鲁棒性分析：PSO 算法变异系数为 87.51%，对参数变化极为敏感，尤其是惯性权重 w；ACO 算法变异系数为 92.76%，展现出最高的参数敏感性，不同参数组合下性能差异巨大；irefly 算法变异系数为 40.35%，相对前两者更稳定，但仍然敏感度较高；osenbrock 函数的特殊地形结构（狭长弯曲山谷）对算法参数配置提出了严格要求。

综合鲁棒性排序：Firefly>ACO>PSO。Firefly 算法在大多数测试函数上展现出最低的参数敏感性，特别是在 Rastrigin 和 Ackley 函数上表现出色。这可能是因为 Firefly 算法基于亮度和吸引力的机制提供了更平滑的搜索过程，对参数变化不那么敏感。PSO 算法虽然在求解精度上表现最佳，但其参数敏感性较高，尤其是在 Ackley 和 Rosenbrock 函数上，这对实际应用中的参数调优提出了更高要求。ACO 算法整体鲁棒性居中，但在 Rosenbrock 函数上表现最差，参数选择对其性能影响显著。

五、 结论与展望

5.1 结论

本研究通过系统比较分析粒子群优化算法（PSO）、蚁群算法（ACO）和萤火虫算法（FA）在不同测试函数上的性能，得出以下结论：

1. PSO 总体表现最佳：在收敛速度、求解精度、计算效率和维度扩展性方面，PSO 都表现出明显优势。这主要得益于其简单有效的速度-位置更新机制和较少的参数调整需求。PSO 特别适合计算资源有限或需要快速获得较好解的场景。

2. ACO 在连续优化问题上劣势明显：虽然 ACO 在组合优化问题中表现出色，但在连续函数优化方面存在实现复杂、收敛慢、计算开销大等问题。然而，ACO 在某些情况下展现出较好的鲁棒性，适合需要稳定解的场景。

3. FA 在多峰函数上有潜力：FA 在处理多峰复杂函数时展现出较好性能，特别是在能够同时定位多个局部最优解方面有独特优势。但其计算复杂度高，参数敏感性强，限制了其应用范围。

4. 参数敏感性分析：三种算法都对参数设置有不同程度的敏感性，其中 PSO 参数调整最为简单，FA 次之，ACO 最为复杂。适当的参数设置对算法性能有显著影响。

5. 算法选择建议：

- 对于计算资源有限或需要快速结果的场景，PSO 是首选；
- 对于多峰复杂函数或需要同时找到多个局部最优解的问题，FA 更优；
- 对于组合优化问题，ACO 仍然具有竞争力；
- 在高维优化问题中，PSO 的优势更为明显。

5.2 展望

基于本研究结果，未来研究可以从以下几个方向展开：

1. 算法混合：结合不同算法的优势，如 PSO 的快速收敛能力与 FA 的多峰处理能力，或 ACO 的全局搜索能力与 PSO 的局部精确搜索能力，开发混合算法提高性能。

2. 自适应参数调整：研究参数的动态自适应调整机制，减少人工调参的工作量，同时提高算法在不同问题上的适应性。

3. 并行实现：探索这些算法的并行实现方案，特别是 FA 和 ACO 这类计算复杂度高的算法，通过并行计算提高效率。

4. 约束处理机制：研究这些算法在处理约束优化问题时的性能和策略改进。

5. 多目标优化扩展：将这些算法扩展到多目标优化领域，评估它们在处理多目标问题时的性能。

6. 实际应用测试：将这些算法应用于实际工程问题，如神经网络参数优化、调度规划、资源分配等，验证其实用性。

本研究为选择和应用群体智能优化算法提供了系统的参考依据，有助于在实际问题中根据具体需求选择合适的优化算法，或者进行算法改进和混合，以获得更好的优化效果。